**Full Stack Development Mini Project:**

**Expense Tracker**

**By Dhruva Sandu – PB55 (PRN:1032211460)**

**Introduction:**

The Expense Tracker application is a web-based platform designed to assist users in managing and tracking their expenses efficiently. The application focuses on providing a user-friendly interface for creating and managing financial events, tracking expenses related to those events, and maintaining a record of past financial activities. By leveraging modern web technologies, the project aims to streamline the process of budgeting and spending.

The primary objective of the Expense Tracker application is to provide users with a comprehensive financial management tool. Users can efficiently track and manage their expenses by creating and organizing events with specified budgets. The application ensures secure user authentication, employs React.js for a dynamic and responsive user interface, utilizes Node.js and Express.js for the backend server, and stores data securely in a MongoDB database. The project aims to streamline budgeting processes, empower users in making informed financial decisions, and maintain a systematic record of their financial history.

**Technologies used in the Project:**

In this mini project, I have utilised the MERN (MongoDB, Express, React, Node) stack to develop both, the frontend and backend parts of the project.

**Frontend (React.js):**

1. **React.js:**

React.js is a JavaScript library for building user interfaces. In this project, React is used for the frontend to create reusable UI components and manage the application's state efficiently.

1. **React Router:**

React Router is used for client-side routing in a React application. It enables navigation between different views/components based on the URL. This project uses it to define routes for different pages, such as login, signup, dashboard, event details, etc.

1. **Redux:**

Redux is a predictable state container for JavaScript apps. In our project, Redux is utilized for state management. The global state is stored in a Redux store, and actions are dispatched to update or retrieve data. This helps manage user authentication and temporary data like the event name and the username.

1. **Axios:**

Axios is a promise-based HTTP client used for making asynchronous requests to the backend API. It is employed in our project to handle communication with the server, fetching and sending data.

**Backend (Node.js, Express.js & MongoDB):**

1. **Node.js:**

Node.js is a JavaScript runtime that allows you to execute JavaScript on the server side. The backend server is built using Node.js, providing a non-blocking, event-driven architecture.

1. **Express.js:**

Express.js is a web application framework for Node.js. It simplifies the process of building robust and scalable web applications. In this project, Express is used to create routes, handle HTTP requests, and manage middleware.

1. **MongoDB:**

MongoDB is a NoSQL database used for storing data. In this project, MongoDB is used to store user information, event details, and expenses. Mongoose, an ODM (Object Data Modeling) library for MongoDB, is used to model the application data and interact with the database.

1. **Mongoose:**

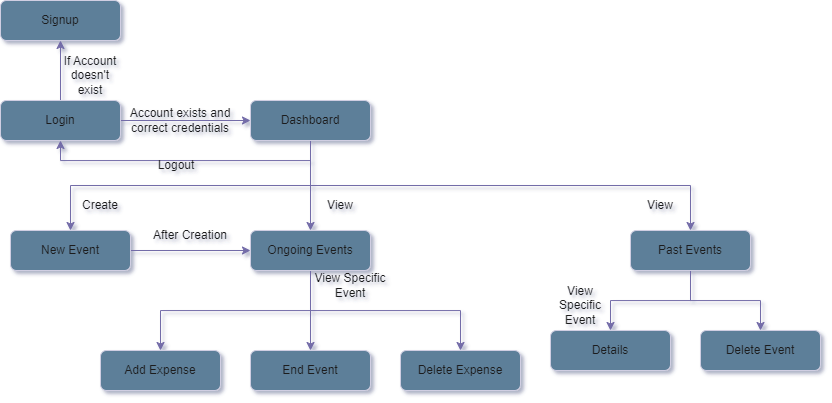
Mongoose is an ODM (Object Data Modeling) library for MongoDB and Node.js. It provides a schema-based solution for modeling application data and interacting with MongoDB.

**Middleware (CORS):**

1. **CORS:**

CORS (Cross-Origin Resource Sharing) is a middleware used in Express to enable cross-origin requests. This is crucial for allowing the frontend, served from a different origin, to communicate with the backend.

**Workflow:**



1. **User Authentication:**

* Users start by either signing up or logging into the application.
* The authentication requests are handled by the Node.js/Express.js server, which communicates with the MongoDB database to verify user credentials.

1. **Dashboard:**

* Upon successful authentication, users are directed to the dashboard.
* The dashboard is implemented using React.js and has navigation to ongoing events, past events and creating new events.

1. **Creating a New Event:**

* Users can create a new event by providing the event name and budget.
* The frontend sends a request to the Node.js/Express.js server, which saves the event details in the MongoDB database.

1. **Ongoing Events:**

* Users can view list of ongoing events and it’s remaining budget and can select an event which they want to update.
* The frontend makes API requests to retrieve ongoing events data.

1. **Event Details:**

* Users can view detailed information about a specific event, including remaining budget and a list of expenses and can also add and delete expenses.
* The React.js frontend communicates with the server to fetch event details and associated expenses.

1. **Adding Expenses:**

* Users can add expenses to a specific event, which updates the remaining budget for that event.
* The frontend sends an API request to add the expense, and the server updates both the event and expense collections in the MongoDB database.

1. **Past Events:**

* Users can view a list of past events.
* The React.js frontend makes API requests to retrieve past events from the server.

1. **Deleting Past Events:**

* Users can delete past events, which triggers the deletion of associated expenses.
* The frontend sends a request to the server, and the server updates the MongoDB database accordingly.

1. **Logging Out:**

* Users have the option to log out, terminating their session.
* The React.js frontend updates the Redux state to clear the user session information.

**Future Scope:**

1. **Enhanced User Authentication and Security:**

Advanced authentication mechanisms, such as multi-factor authentication (MFA) or integration with third-party authentication providers, to enhance user account security.

1. **Real-time Collaboration Features:**

Introducing real-time collaboration features, enabling event participants to share updates, expenses, or messages in real-time. This could involve integrating WebSocket technology or a real-time database.

1. **Expense Analytics and Reporting:**

Implement a feature for users to generate detailed expense reports and analytics for each event. This could involve visualizing spending patterns, providing insights, and helping users make informed financial decisions.

1. **Mobile Application Development:**

Extend the application's reach by developing mobile applications for iOS and Android platforms. Mobile apps can offer a more convenient and accessible experience for users on the go.

1. **Integration with Financial Services or APIs:**

Integration with financial services or APIs to automate certain financial processes. This could include features like automatic expense categorization, integration with digital payment platforms, or even generating financial summaries.

**Conclusion:**

In summary, the project is a full-stack web application using React.js for the frontend, Node.js/Express for the backend, and MongoDB for data storage. It aims to simplify event budget management, focusing on user-friendly interfaces for organizing events and tracking expenses. The workflow involves React.js components interacting with the Node.js backend through RESTful APIs, facilitated by state management with Redux.

The project offers future possibilities such as enhanced user authentication, real-time collaboration features, advanced expense analytics, mobile application development, and financial service integration. These improvements aim to elevate the user experience and provide more sophisticated tools for event planning and financial management. Further details on the project's architecture and technologies will be explored in subsequent sections.

**References:**

<https://www.geeksforgeeks.org/how-to-connect-mongodb-with-reactjs/>

<https://www.geeksforgeeks.org/nodejs/>

<https://stackoverflow.com/>

<https://react-redux.js.org/introduction/getting-started>

<https://www.tutorialspoint.com/expressjs/index.htm>

<https://www.geeksforgeeks.org/mern-stack/>

<https://www.geeksforgeeks.org/know-the-difference-between-rest-api-and-restful-api/>

<https://www.scaler.com/topics/nodejs/cors-in-node-js/>

<https://expressjs.com/en/resources/middleware/cors.html>